



Ressourcesinformatiques

Python 3

Les fondamentaux du langage

Sébastien CHAZALLET

Téléchargement
www.editions-eni.fr



Les éléments à télécharger sont disponibles à l'adresse suivante :
<http://www.editions-eni.fr>
Saisissez la référence ENI de l'ouvrage **RI3PYT** dans la zone de recherche
et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Avant-propos

1. Contenu de l'ouvrage	29
2. Progressivité de l'ouvrage	30
3. À destination des enseignants et élèves	32
4. À destination des chercheurs ou doctorants	33
5. À destination de ceux qui viennent d'un autre langage	34

Partie 1 : Les atouts de Python

Chapitre 1.1

Python dans le paysage informatique

1. Petite histoire des langages informatiques	35
1.1 Informatique théorique	35
1.2 Chronologie de l'informatique	36
1.2.1 Évolutions des problématiques liées à l'informatique	36
1.2.2 Chronologie des langages informatiques	37
1.3 Histoire de Python	42
1.3.1 La genèse	42
1.3.2 Extension du périmètre fonctionnel	43
1.3.3 Évolution de la licence	44
1.3.4 Avenir	44
2. Typologie des langages de programmation	45
2.1 Paradigmes	45
2.1.1 Définition	45
2.1.2 Paradigme impératif et dérivés	46
2.1.3 Paradigme objet et dérivés	46
2.1.4 Programmation orientée aspect	47
2.1.5 Paradigme fonctionnel	47
2.1.6 Paradigme logique	48

2.1.7	Programmation concurrente	48
2.1.8	Synthèse	48
2.2	Interopérabilité	49
2.3	Niveau de programmation	51
2.3.1	Machine	51
2.3.2	Bas niveau	51
2.3.3	Haut niveau	52
2.4	Typage	53
2.4.1	Faible vs fort	53
2.4.2	Statique vs dynamique	53
2.5	Grammaire	53
2.5.1	Langages formels	53
2.5.2	Syntaxe	54
3.	Situer Python dans le paysage informatique	54
3.1	Typologie de Python	54
3.1.1	Grammaire et syntaxe	54
3.1.2	Typage	55
3.1.3	Niveau de programmation	55
3.1.4	Paradigmes supportés	55
3.2	Positionnement stratégique du langage Python	56
3.2.1	Segments de marchés	56
3.2.2	Niveau de complexité	56
3.2.3	Forces du langage	57
3.2.4	Points faibles	57
3.3	Intégration avec d'autres langages	58
3.3.1	Extensions C	58
3.3.2	Intégration de programmes écrits en C	58
3.3.3	Intégration de programmes Python dans du C	58
3.3.4	Intégration de programmes écrits en Java	58
3.3.5	Intégration de programmes Python dans Java	58
3.3.6	Autres intégrations	58

Chapitre 1.2
Présentation de Python

- 1. Philosophie 59
 - 1.1 Python en quelques lignes 59
 - 1.1.1 D'où vient le nom « Python » ? 59
 - 1.1.2 Présentation technique 60
 - 1.1.3 Présentation conceptuelle 60
 - 1.2 Comparaison avec d'autres langages 60
 - 1.2.1 Shell 60
 - 1.2.2 Perl 61
 - 1.2.3 C, C++ 61
 - 1.2.4 Java 63
 - 1.2.5 PHP 64
 - 1.3 Grands principes 66
 - 1.3.1 Le zen de Python 66
 - 1.3.2 Le développeur n'est pas stupide 66
 - 1.3.3 Documentation 67
 - 1.3.4 Python est livré piles incluses 67
 - 1.3.5 Duck Typing 68
 - 1.3.6 Notion de code pythonique 68
- 2. Gouvernance 68
 - 2.1 Développement 68
 - 2.1.1 Branches 68
 - 2.1.2 Communauté 69
 - 2.2 Mode de gouvernance 70
 - 2.2.1 Créateur du langage 70
 - 2.2.2 PEP 70
 - 2.2.3 Prise de décisions 70
- 3. Que contient Python ? 71
 - 3.1 Une grammaire et une syntaxe 71
 - 3.2 Plusieurs implémentations 71
 - 3.3 Une bibliothèque standard 72
 - 3.4 Des bibliothèques tierces 72
 - 3.5 Des frameworks 72

4. Phases d'exécution d'un programme Python.....	73
4.1 Chargement de la machine virtuelle	73
4.2 Compilation	73
4.3 Interprétation	73

Chapitre 1.3

Pourquoi choisir Python

1. Qualités du langage	75
1.1 Couverture fonctionnelle	75
1.2 Ticket d'entrée	76
1.3 Garanties	77
2. Diffusion.....	79
2.1 Entreprises.....	79
2.2 Le monde de la recherche	80
2.3 Le monde de l'éducation.....	81
2.4 Communauté	81
3. Références.....	83
3.1 Poids lourds de l'industrie informatique	83
3.1.1 Google.....	83
3.1.2 Mozilla	84
3.1.3 Microsoft	84
3.1.4 Canonical	84
3.1.5 Cisco.....	85
3.2 Entreprises innovantes	85
3.2.1 Services de stockage en ligne.....	85
3.2.2 Informatique dématérialisée	85
3.2.3 Forge.....	86
3.2.4 Réseaux sociaux.....	86
3.3 Éditeurs de contenus.....	86
3.3.1 Disney Animation Studio	86
3.3.2 YouTube.....	86
3.3.3 Box ADSL	86
3.3.4 Spotify	86
3.4 Éditeurs de logiciels.....	87

- 4. Retours d'expérience 87
 - 4.1 Ressentis 87
 - 4.2 Développement dans un contexte hétérogène 88
 - 4.3 Développement rapide 89
 - 4.4 Vendre des développements Python 89
 - 4.5 Naviguer dans l'inconnu 89

Chapitre 1.4

Installer son environnement de développement

- 1. Installer Python 91
 - 1.1 Windows 91
 - 1.2 Mac OS 94
 - 1.3 UNIX / Linux 94
 - 1.4 Compilation 95
 - 1.5 Compilation de Python 3.4 96
- 2. Installer des bibliothèques externes 96
 - 2.1 Installateur ou gestionnaire de paquets 96
 - 2.2 PIP 96
 - 2.3 Environnement virtuel 97
- 3. Installer un IDE 98
 - 3.1 Console et outils associés 98
 - 3.2 Eclipse + PyDev 98
 - 3.3 Aptana 99
 - 3.4 Eric 99
 - 3.5 PyCharm 99
 - 3.6 Autres solutions 99
 - 3.7 StacklessPython 99
 - 3.8 Environnement hétérogène 100
- 4. Utilisation de la console 100
 - 4.1 Console standard 100
 - 4.2 iPython 100
- 5. Distribuer ses propres applications. 101
 - 5.1 Utilitaire de distribution 101
 - 5.2 Création de paquets distribuables 101
 - 5.3 Distribution binaire pour Windows 102

5.4	Distribution binaire pour Mac	102
5.5	Distribution binaire toute plateforme	102
5.6	Distribution spécifique à un environnement	103

Partie 2 : Les fondamentaux du langage

Chapitre 2.1

Algorithmique de base

1.	Délimiteurs	105
1.1	Instruction	105
1.2	Une ligne de code = une instruction	105
1.3	Commentaire	106
1.4	Une instruction sur plusieurs lignes	106
1.5	Mots-clés	107
1.6	Mots réservés	107
1.7	Indentation	108
1.8	Symboles	109
1.9	Opérateurs	113
1.10	Utilisation du caractère souligné	116
1.11	PEP-8	117
1.12	PEP-7	117
1.13	PEP-257	118
2.	Instructions	118
2.1	Définitions	118
2.1.1	Variable	118
2.1.2	Fonction	119
2.1.3	Fonctions lambda	121
2.1.4	Classe	121
2.1.5	Instruction vide	122
2.1.6	Suppression	123
2.1.7	Renvoyer le résultat de la fonction	124
2.2	Instructions conditionnelles	125
2.2.1	Définition	125
2.2.2	Condition	125
2.2.3	Instruction if	125
2.2.4	Instruction elif	126

- 2.2.5 Instruction else 126
- 2.2.6 Instruction switch 128
- 2.2.7 Interruptions 128
- 2.2.8 Approfondissement des conditions 128
- 2.2.9 Performances 129
- 2.3 Itérations 130
 - 2.3.1 Instruction for 130
 - 2.3.2 Instruction while 131
 - 2.3.3 Quelle différence entre for et while 132
 - 2.3.4 Instruction break 132
 - 2.3.5 Instruction return 134
 - 2.3.6 Instruction continue 134
 - 2.3.7 Instruction else 134
 - 2.3.8 Générateurs 136
- 2.4 Constructions fonctionnelles 139
 - 2.4.1 Générateurs 139
 - 2.4.2 Compréhensions de listes 139
 - 2.4.3 Compréhensions d'ensembles 139
 - 2.4.4 Compréhensions de dictionnaires 139
- 2.5 Gestion des exceptions 139
 - 2.5.1 Présentation rapide des exceptions 139
 - 2.5.2 Lever une exception 140
 - 2.5.3 Pourquoi lever une exception ? 141
 - 2.5.4 Assertions 142
 - 2.5.5 Capturer une exception 143
 - 2.5.6 Effectuer un traitement de l'exception 144
 - 2.5.7 Gérer la sortie du bloc de capture 146
 - 2.5.8 Gérer le non-déclenchement d'exceptions 147
 - 2.5.9 Prise et libération de ressources 148
- 2.6 Divers 149
 - 2.6.1 Gérer des imports 149
 - 2.6.2 Traverser les espaces de nommage 150
 - 2.6.3 Fonctions print, help, eval et exec 152

Chapitre 2.2

Déclarations

1. Variable.....	155
1.1 Qu'est-ce qu'une variable ?.....	155
1.1.1 Contenu	155
1.1.2 Contenant	155
1.1.3 Modes de modification d'une variable	157
1.2 Typage dynamique	161
1.2.1 Affectation : rappels.....	161
1.2.2 Primitive type et nature du type.....	161
1.2.3 Caractéristiques du typage Python.....	162
1.3 Visibilité.....	164
1.3.1 Espace global.....	164
1.3.2 Notion de bloc	165
2. Fonction	169
2.1 Déclaration	169
2.2 Paramètres.....	170
2.2.1 Signature d'une fonction.....	170
2.2.2 Notion d'argument ou de paramètre	171
2.2.3 Valeur par défaut	171
2.2.4 Paramètres nommés.....	173
2.2.5 Déclaration de paramètres extensibles	174
2.2.6 Passage de paramètres étoilés	175
2.2.7 Signature universelle	176
2.2.8 Obliger un paramètre à être nommé (keyword-only)	177
2.2.9 Annotations	179
3. Classe	182
3.1 Déclaration	182
3.1.1 Signature.....	182
3.1.2 Attribut.....	183
3.1.3 Méthode	183
3.1.4 Bloc local.....	184
3.2 Instanciation	185
3.2.1 Syntaxe.....	185
3.2.2 Relation entre l'instance et la classe	185

- 4. Module 186
 - 4.1 Déclaration 186
 - 4.2 Instructions spécifiques 187
 - 4.3 Comment appréhender le contenu d'un module ? 187

Chapitre 2.3
Modèle objet

- 1. Tout est objet 189
 - 1.1 Principes 189
 - 1.1.1 Quel sens donner à « objet » ? 189
 - 1.1.2 Adaptation de la théorie objet dans Python 190
 - 1.1.3 Généralités 191
 - 1.2 Classes 192
 - 1.2.1 Introduction 192
 - 1.2.2 Déclaration impérative d'une classe 192
 - 1.2.3 Instance 193
 - 1.2.4 Objet courant 195
 - 1.2.5 Déclaration par prototype d'une classe 195
 - 1.3 Méthodes 197
 - 1.3.1 Déclaration 197
 - 1.3.2 Appel de méthode 199
 - 1.3.3 Méthodes et attributs spéciaux 202
 - 1.3.4 Constructeur et initialisateur 206
 - 1.3.5 Gestion automatisée des attributs 207
 - 1.3.6 Intérêt du paradigme objet 208
 - 1.3.7 Relation entre objets 208
 - 1.4 Héritage 209
 - 1.4.1 Polymorphisme par sous-typage 209
 - 1.4.2 Surcharge de méthode 210
 - 1.4.3 Surcharge des opérateurs 212
 - 1.4.4 Polymorphisme paramétrique 213
 - 1.4.5 Héritage multiple 215
- 2. Autres outils de la programmation objet 217
 - 2.1 Principes 217
 - 2.2 Interfaces 218

2.3	Attributs	220
2.4	Propriétés	223
2.5	Emplacements	226
2.6	Métaclasses	227
2.7	Classes abstraites	230
2.8	La Zope Component Architecture	233
2.8.1	Présentation	233
2.8.2	Installation	234
2.8.3	Définir une interface et un composant	234
2.8.4	Autres fonctionnalités	235
2.8.5	Avantages de la ZCA	235
3.	Fonctions spéciales et primitives associées	236
3.1	Personnalisation	236
3.1.1	Classes	236
3.1.2	Instances	238
3.1.3	Comparaison	239
3.1.4	Évaluation booléenne	239
3.1.5	Relations d'héritage ou de classe à instance	240
3.2	Classes particulières	241
3.2.1	Itérateurs	241
3.2.2	Conteneurs	243
3.2.3	Instances assimilables à des fonctions	243
3.2.4	Ressources à protéger	244
3.2.5	Types	245

Chapitre 2.4

Types de données et algorithmes appliqués

1.	Nombres	247
1.1	Types	247
1.1.1	Entiers	247
1.1.2	Réels	248
1.1.3	Socle commun aux nombres entiers et réels	249
1.1.4	Méthodes dédiées aux nombres entiers	251
1.1.5	Méthodes dédiées aux nombres réels	251
1.1.6	Complexes	252

1.2	La console Python, la calculatrice par excellence	253
1.2.1	Opérateurs mathématiques binaires	253
1.2.2	Opérateurs binaires particuliers	254
1.2.3	Opérateurs mathématiques unaires	255
1.2.4	Arrondis	256
1.2.5	Opérateurs de comparaison	259
1.2.6	Opérations mathématiques n-aires	260
1.2.7	Fonctions mathématiques usuelles.	261
1.3	Représentations d'un nombre	267
1.3.1	Représentation décimale	267
1.3.2	Représentation par un exposant.	267
1.3.3	Représentation par une fraction.	268
1.3.4	Représentation hexadécimale	268
1.3.5	Représentation octale	270
1.3.6	Représentation binaire	271
1.3.7	Opérations binaires	271
1.3.8	Longueur de la représentation mémoire d'un entier	273
1.4	Conversions	274
1.4.1	Conversion entre entiers et réels	274
1.4.2	Conversion entre réels et complexes	275
1.4.3	Conversion vers un booléen	276
1.5	Travailler avec des variables	277
1.5.1	Un nombre est non mutable.	277
1.5.2	Modifier la valeur d'une variable	277
1.5.3	Opérateurs d'incrémentaion	278
1.6	Statistiques	279
1.7	Calcul scientifique	281
1.7.1	Le calcul scientifique, pour quoi faire ?	281
1.7.2	Python, une alternative libre et crédible	281
1.7.3	Quelques bibliothèques.	282
2.	Séquences.	282
2.1	Présentation des différents types de séquences	282
2.1.1	Généralités	282
2.1.2	Les listes	283
2.1.3	Les n-uplets	285
2.1.4	Conversion entre listes et n-uplets	287
2.1.5	Socle commun entre liste et n-uplet	287

2.1.6	Notion d'itérateur	288
2.2	Utilisation des indices et des tranches	291
2.2.1	Définition de l'indice d'un objet et des occurrences	291
2.2.2	Utiliser l'indice pour adresser la séquence	292
2.2.3	Retrouver les occurrences d'un objet et leurs indices	293
2.2.4	Taille d'une liste, comptage d'occurrences	295
2.2.5	Utiliser l'indice pour modifier ou supprimer	296
2.2.6	Itération simple	298
2.2.7	Présentation de la notion de tranches (slices)	302
2.2.8	Cas particulier de la branche 2.x de Python	312
2.2.9	Utilisation basique des tranches	313
2.2.10	Utilisation avancée des tranches	314
2.3	Utilisation des opérateurs	316
2.3.1	Opérateur +	316
2.3.2	Opérateur *	318
2.3.3	Opérateur +=	320
2.3.4	Opérateur *=	321
2.3.5	Opérateur in	323
2.3.6	Opérateurs de comparaison	324
2.4	Méthodes de modifications	325
2.4.1	Ajouter des éléments dans une liste et un n-uplet	325
2.4.2	Supprimer un objet d'une liste et d'un n-uplet	328
2.4.3	Solutions de contournement pour la modification de n-uplets	332
2.4.4	Renverser une liste ou un tuple	333
2.4.5	Trier une liste	334
2.5	Utilisation avancée des listes	337
2.5.1	Opérations d'ensemble	337
2.5.2	Pivoter une séquence	338
2.5.3	Itérer correctement	339
2.5.4	Programmation fonctionnelle	340
2.5.5	Compréhensions de listes	343
2.5.6	Itérations avancées	344
2.5.7	Combinatoire	349
2.6	Adapter les listes à des besoins spécifiques	352
2.6.1	Liste d'entiers	352
2.6.2	Présentation du type array	354

2.6.3	Utiliser une liste comme pile	356
2.6.4	Utiliser une liste comme file d'attente	356
2.6.5	Utiliser des listes pour représenter des matrices	358
2.6.6	Liste sans doublons	359
2.7	Autres types de données.	361
3.	Ensembles.	364
3.1	Présentation	364
3.1.1	Définition d'un ensemble	364
3.1.2	Différences entre set et frozenset	365
3.1.3	Utilisation pour dédoublonner des listes	366
3.1.4	Rajouter une relation d'ordre	366
3.2	Opérations ensemblistes	367
3.2.1	Opérateurs pour un ensemble à partir de deux autres	367
3.2.2	Opérateurs pour modifier un ensemble à partir d'un autre	368
3.2.3	Méthodes équivalentes à la création ou modification ensembliste.	369
3.2.4	Méthodes de comparaison des ensembles	369
3.2.5	Exemples non classiques d'utilisation	371
3.3	Méthodes de modification d'un ensemble	374
3.3.1	Ajouter un élément	374
3.3.2	Supprimer un élément	375
3.3.3	Vider un ensemble	376
3.3.4	Dupliquer un élément	376
3.3.5	Sortir une valeur d'un ensemble	377
3.3.6	Utiliser un ensemble comme un recycleur d'objets.	378
3.3.7	Algorithmique avancée : résolution du problème des n-dames	381
4.	Chaînes de caractères	383
4.1	Présentation	383
4.1.1	Définition.	383
4.1.2	Vocabulaire.	384
4.1.3	Spécificités de la branche 2.x	385
4.1.4	Changements apportés par la branche 3.x.	386
4.1.5	Chaîne de caractères en tant que séquence de caractères	389
4.1.6	Caractères.	391
4.1.7	Opérateurs de comparaison	392

4.2	Formatage de chaînes de caractères	395
4.2.1	Opérateur modulo	395
4.2.2	Méthodes de formatage sur l'ensemble de la chaîne	401
4.2.3	Nouvelle méthode de formatage des variables dans une chaîne	403
4.3	Opérations d'ensemble	407
4.3.1	Séquençage de chaînes	407
4.3.2	Opérations sur la casse	410
4.3.3	Recherche sur une chaîne de caractères	411
4.3.4	Informations sur les caractères	412
4.4	Problématiques relatives à l'encodage	414
4.4.1	Encodage par défaut	414
4.4.2	Encodage du système	414
4.4.3	L'unicode, référence absolue	414
4.4.4	Autres encodages	416
4.4.5	Ponts entre l'unicode et le reste du monde	416
4.4.6	Revenir vers l'unicode	418
4.5	Manipulations de bas niveau avancées	419
4.5.1	Opérations de comptage	419
4.5.2	Une chaîne de caractères vue comme une liste	420
4.5.3	Une chaîne de caractères vue comme un ensemble de caractères	421
4.6	Représentation mémoire	421
4.6.1	Présentation du type bytes	421
4.6.2	Lien avec les chaînes de caractères	422
4.6.3	Présentation du type bytearray	424
4.6.4	Gestion d'un jeu de caractères	425
5.	Dictionnaires	431
5.1	Présentation	431
5.1.1	Définition	431
5.1.2	Évolutions et différences entre les branches 2.x et 3.x	432
5.1.3	Vues de dictionnaires	433
5.1.4	Instanciation	436
5.1.5	Compréhension de dictionnaire	436
5.2	Manipuler un dictionnaire	437
5.2.1	Récupérer une valeur d'un dictionnaire	437
5.2.2	Modifier les valeurs d'un dictionnaire	438

5.2.3	Supprimer une entrée d'un dictionnaire	439
5.2.4	Dupliquer un dictionnaire	439
5.2.5	Utiliser le dictionnaire comme agrégateur de données	440
5.2.6	Méthodes d'itération	441
5.3	Utilisation avancée des dictionnaires	442
5.3.1	Rajouter une relation d'ordre	442
5.3.2	Algorithmique classique	446
5.3.3	Adapter les dictionnaires à des besoins spécifiques	447
5.3.4	Représentation universelle de données	448
6.	Booléens	449
6.1	Le type booléen	449
6.1.1	Classe bool	449
6.1.2	Les deux objets True et False	450
6.1.3	Différence entre l'opérateur d'égalité et d'identité	450
6.2	Évaluation booléenne	451
6.2.1	Méthode générique	451
6.2.2	Objets classiques	451
7.	Données temporelles	452
7.1	Gérer une date calendaire	452
7.1.1	Notion de date calendaire	452
7.1.2	Travailler sur une date	453
7.1.3	Considérations astronomiques	453
7.1.4	Considérations historiques	454
7.1.5	Considérations techniques	454
7.1.6	Représentation textuelle	455
7.2	Gérer un horaire ou un moment d'une journée	457
7.2.1	Notion d'instant	457
7.2.2	Notion de fuseau horaire	458
7.2.3	Représentation textuelle	459
7.3	Gérer un instant absolu	460
7.3.1	Notion d'instant absolu	460
7.3.2	Rapport avec les notions précédentes	460
7.3.3	Représentation textuelle	462
7.3.4	Gestion des fuseaux horaires	463
7.3.5	Créer une date à partir d'une représentation textuelle	463

7.4	Gérer une différence entre deux dates ou instants	463
7.4.1	Notion de différence et de résolution	463
7.4.2	Considérations techniques	465
7.4.3	Utilisation avec des dates calendaires	466
7.4.4	Utilisation avec des horaires	466
7.4.5	Utilisation avec des dates absolues	466
7.4.6	La seconde comme unité de base	467
7.5	Spécificités des fuseaux horaires	467
7.6	Problématiques de bas niveau	468
7.6.1	Timestamp et struct_time	468
7.6.2	Mesures de performances	469
7.7	Utilisation du calendrier	472
7.7.1	Présentation du module calendar	472
7.7.2	Fonctions essentielles du calendrier	477

Chapitre 2.5

Motifs de conception

1.	Définition	479
1.1	Positionnement par rapport à la notion d'objet	479
1.2	Organisation du chapitre	480
1.3	Positionnement par rapport à d'autres concepts	481
2.	Création	481
2.1	Singleton	481
2.2	Fabrique	482
2.2.1	Présentation de la problématique	482
2.2.2	Solutions	482
2.2.3	Conséquences	484
2.3	Fabrique abstraite	485
2.3.1	Présentation de la problématique	485
2.3.2	Solution	485
2.4	Monteur	485
2.4.1	Présentation de la problématique	485
2.4.2	Solution	486
2.4.3	Conséquences	488

- 2.5 Prototype 488
 - 2.5.1 Présentation de la problématique 488
 - 2.5.2 Solution 488
 - 2.5.3 Conséquences. 491
- 3. Structuration 491
 - 3.1 Adaptateur 491
 - 3.1.1 Présentation de la problématique 491
 - 3.1.2 Solution 491
 - 3.1.3 Conséquences. 493
 - 3.2 Pont 494
 - 3.2.1 Présentation de la problématique 494
 - 3.2.2 Solution 495
 - 3.2.3 Conséquences. 497
 - 3.3 Composite 498
 - 3.3.1 Présentation de la problématique 498
 - 3.3.2 Solution 498
 - 3.3.3 Conséquences. 500
 - 3.4 Décorateur 500
 - 3.4.1 Présentation de la problématique 500
 - 3.4.2 Solution 500
 - 3.4.3 Conséquences. 502
 - 3.5 Façade 503
 - 3.5.1 Présentation de la problématique 503
 - 3.5.2 Solution 503
 - 3.5.3 Conséquences. 505
 - 3.6 Poids-mouche 505
 - 3.6.1 Présentation de la problématique 505
 - 3.6.2 Solution 505
 - 3.6.3 Conséquences. 506
 - 3.7 Proxy 507
 - 3.7.1 Présentation de la problématique 507
 - 3.7.2 Solution 507
 - 3.7.3 Conséquences. 509

4.	Comportement	509
4.1	Chaîne de responsabilité	509
4.1.1	Présentation de la problématique	509
4.1.2	Solution	509
4.1.3	Conséquences	510
4.2	Commande	510
4.2.1	Présentation de la problématique	510
4.2.2	Solution	511
4.2.3	Conséquences	512
4.3	Itérateur	513
4.3.1	Présentation de la problématique	513
4.3.2	Solution	513
4.3.3	Conséquences	515
4.4	Memento	515
4.4.1	Présentation de la problématique	515
4.4.2	Solution	516
4.4.3	Conséquences	517
4.5	Visiteur	517
4.5.1	Présentation de la problématique	517
4.5.2	Solution	517
4.5.3	Conséquences	518
4.6	Observateur	518
4.6.1	Présentation de la problématique	518
4.6.2	Solution	518
4.6.3	Conséquences	519
4.7	Stratégie	519
4.7.1	Présentation de la problématique	519
4.7.2	Solution	519
4.7.3	Conséquences	520
4.8	Fonction de rappel	520
4.8.1	Présentation de la problématique	520
4.8.2	Solution	520
4.8.3	Conséquences	521

- 5. ZCA 521
 - 5.1 Rappels 521
 - 5.2 Adaptateur 522
 - 5.2.1 Déclaration..... 522
 - 5.2.2 Utilisation 523
 - 5.3 Utilitaire 524
 - 5.3.1 Déclaration..... 524
 - 5.3.2 Utilisation 525
 - 5.4 Fabrique..... 525
 - 5.4.1 Déclaration..... 525
 - 5.4.2 Utilisation 526
 - 5.5 Pour aller plus loin 526

Partie 3 : Les fonctionnalités

Chapitre 3.1

Manipulation de données

- 1. Bases de données 527
 - 1.1 Présentation 527
 - 1.2 Accès à une base de données relationnelle 528
 - 1.2.1 Point d'entrée..... 528
 - 1.2.2 MySQL..... 528
 - 1.2.3 PostgreSQL..... 533
 - 1.2.4 SQLite..... 536
 - 1.2.5 Oracle 536
 - 1.3 Utilisation d'un ORM 537
 - 1.3.1 Qu'est-ce qu'un ORM ? 537
 - 1.3.2 ORM proposés par Python 537
 - 1.3.3 SQLAlchemy 538
 - 1.4 Autres bases de données..... 545
 - 1.4.1 CSV..... 545
 - 1.4.2 NoSQL 553
 - 1.4.3 Base de données orientée objet : ZODB..... 553
 - 1.4.4 Base de données de type clé-valeur : REDIS..... 558
 - 1.4.5 Bases de données orientées documents :
 CouchDB et MongoDB 560

2.	LDAP	561
2.1	Présentation	561
2.1.1	Protocole	561
2.1.2	Serveurs	561
2.1.3	Terminologie	562
2.2	Installation	562
2.3	Ouvrir une connexion à un serveur	563
2.4	Effectuer une recherche	564
2.5	Synchrone vs asynchrone	565
2.6	Connexions sécurisées	566
3.	XML	567
3.1	XML et les technologies qui gravitent autour	567
3.1.1	Définition de XML, terminologie associée	567
3.1.2	Notion de schéma	568
3.1.3	Avantages et inconvénients de XML	569
3.1.4	Différentes manières de parcourir un fichier XML	570
3.1.5	Modules Python dédiés au XML	571
3.2	Valider un document XML	571
3.2.1	Document XML	571
3.2.2	Schéma DTD	573
3.2.3	Schéma XSD	573
3.2.4	Schéma RNG (RelaxNG)	574
3.2.5	Schematron	574
3.3	DOM	575
3.3.1	Lecture	575
3.3.2	Écriture	576
3.4	SAX	578
3.4.1	Support de SAX dans lxml	578
3.4.2	API SAX Allégée	579
3.5	XPath	580
3.6	XSLT	583
3.7	Cas spécifiques des fichiers HTML	584
3.7.1	Problématique	584
3.7.2	Parser un fichier HTML à la façon DOM	585
3.7.3	Parser un fichier HTML à la façon SAX	586

- 4. Outils de manipulation de données 588
 - 4.1 Encrypter une donnée. 588
 - 4.1.1 Fonctions de hachage 588
 - 4.1.2 Code d’authentification de message. 590
 - 4.1.3 Stéganographie. 591
 - 4.2 Générer des nombres aléatoires 595
 - 4.3 Expressions régulières. 596
- 5. Travailler avec des médias. 601
 - 5.1 Images 601
 - 5.1.1 Représentation informatique d’une image. 601
 - 5.1.2 Présentation de Pillow. 602
 - 5.1.3 Formats d’images matricielles. 603
 - 5.1.4 Récupérer des informations d’une image. 606
 - 5.1.5 Opérations d’ensemble sur une image 607
 - 5.1.6 Travailler avec les calques ou les pixels 609

Chapitre 3.2
Génération de contenu

- 1. PDF 613
 - 1.1 Présentation 613
 - 1.1.1 Format PDF 613
 - 1.1.2 Avantages. 613
 - 1.1.3 Inconvénients. 614
 - 1.1.4 Présentation de la bibliothèque libre 614
 - 1.2 Bas niveau 614
 - 1.2.1 Bibliothèque de données 614
 - 1.2.2 Canvas 617
 - 1.3 Haut niveau. 618
 - 1.3.1 Styles 618
 - 1.3.2 Flux de données 620
 - 1.3.3 Création d’un visuel 622
 - 1.3.4 Template de page. 623
 - 1.3.5 Page contenant plusieurs zones 624

2. OpenDocument	627
2.1 Installation	627
2.2 OpenDocument Texte	627
2.2.1 Hello World	627
2.3 OpenDocument Tableur	627
2.3.1 Principes généraux par rapport au texte	627
2.3.2 Aller plus loin	628

Chapitre 3.3

Programmation parallèle

1. Terminologie	629
1.1 Processus	629
1.2 Tâche	630
2. Utilisation d'une tâche	630
2.1 Gestion d'une tâche	630
2.1.1 Présentation	630
2.1.2 Création	631
2.2 Gestion de plusieurs tâches	634
2.2.1 Lancement et contrôle	634
2.2.2 Opportunité d'utiliser une tâche	636
2.3 Résolution des problématiques liées	638
2.3.1 Synchronisation	638
2.3.2 Synchronisation conditionnelle	641
2.3.3 Sémaphore	643
3. Utilisation de processus	645
3.1 Gestion d'un processus	645
3.1.1 Présentation	645
3.1.2 Création	646
3.2 Gestion de plusieurs processus	649
3.2.1 Synchronisation	649
3.2.2 Paralléliser un travail	649
3.3 Résolution des problématiques liées	652
3.3.1 Communication interprocessus	652
3.3.2 Partage de données entre processus	653
3.4 Opportunité d'utiliser les processus	654
3.5 Démon	655

- 4. Exécution asynchrone 657
 - 4.1 Introduction 657
 - 4.2 Présentation 658
 - 4.3 Programmation asynchrone 664

Chapitre 3.4
Programmation système et réseau

- 1. Présentation 667
 - 1.1 Définition 667
 - 1.2 Objectifs du chapitre 668
- 2. Écrire des scripts système 668
 - 2.1 Appréhender son système d'exploitation 668
 - 2.1.1 Avertissement 668
 - 2.1.2 Système d'exploitation 668
 - 2.1.3 Processus courant 669
 - 2.1.4 Utilisateurs et groupes 670
 - 2.1.5 Constantes pour le système de fichiers 672
 - 2.1.6 Gérer les chemins 673
 - 2.2 Gestion d'un fichier 674
 - 2.2.1 Ouvrir un fichier 674
 - 2.2.2 Lire un fichier 675
 - 2.2.3 Écrire un fichier 676
 - 2.2.4 Changer les droits d'un fichier 677
 - 2.2.5 Changer de propriétaire ou de groupe 679
 - 2.2.6 Récupérer des informations sur un fichier 680
 - 2.2.7 Supprimer un fichier 681
 - 2.3 Alternatives simples à des commandes bash usuelles 681
 - 2.3.1 Répertoires 681
 - 2.3.2 Fichiers 683
 - 2.3.3 Module de haut niveau 684
 - 2.3.4 Recherche d'un fichier 687
 - 2.4 Exécuter des commandes externes 687
 - 2.4.1 Exécuter et afficher le résultat 687
 - 2.4.2 Exécuter et récupérer le résultat 688

2.5	Utilitaires	689
2.5.1	Différentiel entre fichiers	689
2.5.2	Utilitaire de sauvegarde	692
2.5.3	Lire un fichier de configuration	692
2.5.4	Pickle	693
2.6	Compresser et décompresser un fichier	696
2.6.1	Tarfile	696
2.6.2	Gzip	698
2.6.3	Bz2	699
2.6.4	Zipfile	699
2.6.5	Interface de haut niveau	702
3.	Travailler avec des arguments	703
3.1	Présentation	703
3.2	Mise en œuvre	704
4.	Programmation réseau	708
4.1	Écrire un serveur et un client	708
4.1.1	Utilisation d'une socket TCP	708
4.1.2	Utilisation d'une socket UDP	712
4.1.3	Création d'un serveur TCP	715
4.1.4	Création d'un serveur UDP	717
4.1.5	Un peu plus loin sur le sujet	717
4.2	Utiliser un protocole standard	719
4.2.1	HTTP	719
4.2.2	Proxy	723
4.2.3	Cookies	724
4.2.4	FTP et SFTP	724
4.2.5	SSH	727
4.2.6	POP et POP3	729
4.2.7	IMAP et IMAPS	730
4.2.8	SMTP et SMTPS	732
4.2.9	NNTP	736
4.2.10	IRC	737
4.3	Services web	741
4.3.1	REST	741
4.3.2	SOAP	742
4.3.3	Pyro	744

- 5. Utilisation du matériel 745
 - 5.1 Wake-on-LAN 745
 - 5.1.1 Pré-requis 745
 - 5.1.2 Mise en œuvre 746
 - 5.2 Utilisation du port série 746

Chapitre 3.5
Bonnes pratiques

- 1. Programmation dirigée par les tests 749
 - 1.1 Tests unitaires 749
 - 1.1.1 Principes 749
 - 1.1.2 Interprétation 750
 - 1.1.3 Couverture 751
 - 1.1.4 Outils 752
 - 1.2 Tests de non-régression 754
 - 1.2.1 Actions de développement 754
 - 1.2.2 Gestion de la découverte d'une anomalie par une MOA 754
 - 1.3 Tests fonctionnels 755
 - 1.4 Tests de performance 756
 - 1.5 Intégration continue 759
- 2. Programmation dirigée par la documentation 760
 - 2.1 Documentation interne 760
 - 2.1.1 À destination des développeurs 760
 - 2.1.2 À destination des utilisateurs 761
 - 2.2 Documentation externe 761
 - 2.2.1 Présentation 761
 - 2.2.2 Démarrage rapide 762
 - 2.2.3 Résultat 764
- 3. Optimisation 765
 - 3.1 Qualimétrie 765
 - 3.2 Outils de débogage 767
 - 3.3 Outils de profilage 768

3.4 Règles d'optimisation	769
3.4.1 Pourquoi optimiser ?	769
3.4.2 Règles générales	770
3.4.3 Profiler un algorithme	771
3.4.4 Optimiser l'utilisation de la mémoire	781

Partie 4 : Mise en pratique

Chapitre 4.1

Créer une application web en 30 minutes

1. Description de l'application à construire	785
2. Mise en place	786
2.1 Isolation de l'environnement	786
2.2 Création du projet	787
2.3 Paramétrage	788
2.4 Premiers essais	789
3. Réalisation de l'application	789
3.1 Modèles	789
3.2 Vues	792
3.3 Contrôleurs	794
4. Pour aller plus loin	799

Chapitre 4.2

Créer une application console en 10 minutes

1. Objectif	801
2. Enregistrer le script	802
3. Création des données	802
4. Parseur d'arguments	803

Chapitre 4.3
Créer une application graphique en 20 minutes

- 1. Objectif 805
 - 1.1 Fonctionnel 805
 - 1.2 Technique 805
- 2. Présentation rapide de Gtk et d’astuces 806
 - 2.1 Présentation 806
 - 2.2 Astuces 806
- 3. Démarrer le programme 808
- 4. Interface graphique avec Glade 810
- 5. Créer le composant graphique 813
- 6. Contrôleur 815
- 7. Autres bibliothèques graphiques 816
 - 7.1 TkInter 816
 - 7.2 wxPython 816
 - 7.3 PyQt 817
 - 7.4 PySide 817
 - 7.5 Autres 817

Chapitre 4.4
Créer un jeu en 30 minutes avec PyGame

- 1. Présentation de PyGame 819
- 2. Réalisation d’un jeu Tetris 821
 - 2.1 Présentation du jeu 821
 - 2.2 Présentation des problématiques 821
 - 2.3 Création des constantes 822

Annexes

1. Table UNICODE.....	837
1.1 Script	837
2. Bytes	838
2.1 Script	838
2.2 Résultat	838
Index	843

Chapitre 3.4

Programmation système et réseau

1. Présentation

1.1 Définition

La programmation système se définit par opposition à la programmation d'applications. Il ne s'agit pas de concevoir un logiciel qui va utiliser le système et ses ressources pour effectuer une action, mais de concevoir une des briques qui va s'intégrer au système lui-même. Cela peut donc être le développement d'un pilote pour un matériel, d'une interface réseau ou encore la gestion des ressources.

Par extension, la création d'un programme qui utilise d'autres programmes systèmes est de la programmation système. Le terme programmation système s'étend alors à tout ce qui peut permettre à un administrateur système de résoudre les problématiques usuelles concernant son domaine de compétence, à savoir la gestion des utilisateurs, des périphériques, des processus, de la sauvegarde...

Ainsi, par extension, l'utilisation des commandes **bash** est de la programmation système. L'utilisation des commandes **mysql** et **mysqldump** pour opérer des actions de sauvegarde quotidiennes l'est également.

Un système d'exploitation moderne est écrit majoritairement en C, le reste étant de l'assembleur spécifique à la machine. Ce même système d'exploitation – moderne – a des fonctionnalités de haut niveau, telles que le gestionnaire de paquets qui a besoin par exemple d'effectuer des opérations diverses telles que des échanges réseau pour télécharger des paquets, vérifier leur intégrité, les ouvrir, les installer.

Celles-ci sont le plus souvent écrites en Python, parce qu'il est un langage facile à manipuler, efficace, complet et surtout fiable, disposant de l'outillage nécessaire.

1.2 Objectifs du chapitre

Dans ce chapitre sont présentés les moyens mis à disposition par Python pour permettre d'exécuter des commandes système de manière à effectuer des opérations de maintenance, l'utilisation du système de fichiers, les moyens permettant à Python d'être une alternative crédible à Bash, en particulier par le traitement du passage d'arguments.

Les problématiques liées à la gestion des protocoles réseau, souvent associées à de la programmation système, sont également présentées, et orientées vers la communication entre différents types de clients et de serveurs et différentes technologies. Cela couvre également les services web.

La gestion des tâches et processus à haut niveau est également une problématique faisant partie de la programmation système. Mais étant de haut niveau, elle est utilisée beaucoup plus pour des applications qu'en tant qu'élément de programmation système pure. C'est la raison pour laquelle cette partie est traitée dans un chapitre dédié, le chapitre Programmation parallèle.

2. Écrire des scripts système

2.1 Appréhender son système d'exploitation

2.1.1 Avertissement

L'exécution de commandes externes est intimement liée au système sur lequel se trouve installé Python. D'une part, chaque système d'exploitation possède ses propres commandes. Par exemple, pour lister un répertoire, on utilisera `ls` ou `dir`.

Cette section traite principalement les commandes Unix.

Au-delà des commandes système classiques, certaines commandes comme `mysql` ou `mysqldump` ne peuvent être utilisées que si les programmes adéquats ont été installés, quel que soit le système.

2.1.2 Système d'exploitation

Python propose un module de bas niveau permettant de gérer des informations sur le système d'exploitation :

```
■ >>> import os
```

Voici les deux principaux moyens de vérifier la nature du système :

```
■ >>> os.name  
'posix'  
>>> os.uname ()
```

```

('Linux', 'nom_donne_au_host', '2.6.38-11-generic', '#50-Ubuntu
 SMP Mon Sep 12 21:17:25 UTC 2011', 'x86_64')

```

La première commande donne une standardisation de l'environnement et la seconde des détails sur le nom du système d'exploitation, de la machine, le nom du noyau et sa version ainsi que l'architecture de la machine. Tester ces valeurs permet d'effectuer des choix et d'adapter une application à un environnement précis pour certaines opérations qui le nécessitent.

Voici comment trouver la liste des variables d'environnement :

```
>>> list(os.environ.keys())
```

Et voici comment aller chercher la valeur d'une de ces variables :

```
>>> os.getenv('LANGUAGE')
'fr_FR:fr'
```

L'exploitation de ces variables d'environnement permet également de diriger des choix permettant l'adaptation de l'application. Dans le cas qui vient d'être vu, le choix de la locale peut servir à produire une interface adaptée au langage de l'utilisateur. Il est possible de lire les variables d'environnement sous forme d'octets avec **os.environb** (utile lorsque Python est unicode, mais pas le système).

Python permet également de modifier ces variables d'environnement à l'aide de la méthode **putenv**. L'environnement est alors affecté dans tous les sous-processus.

2.1.3 Processus courant

Python permet d'obtenir des informations sur le processus courant. Ces fonctionnalités ne sont disponibles que pour Linux.

La principale d'entre elles est l'identifiant du processus courant et celui du parent :

```
>>> os.getpid()
5256
>>> os.getppid()
3293
```

Le parent peut correspondre à l'identifiant d'un processus Python père ou à celui de la console lorsque Python est lancé en mode console depuis la console.

Il peut récupérer l'utilisateur affecté au processus et l'utilisateur effectif :

```
>>> os.getuid()
1000
>>> os.geteuid()
1000
```

On peut également avoir des informations textuelles :

```
>>> os.getlogin()
'sch'
```

Et des informations sur les groupes rattachés au processus :

```
>>> os.getgroups()
[4, 20, 24, 46, 112, 120, 122, 1000]
```

Ainsi que sur le 3-uplet (utilisateur courant, effectif, sauvegardé) :

```
>>> os.getresuid()
(1000, 1000, 1000)
```

Voici le 3-uplet des groupes associés (courant, effectif, sauvegardé) :

```
>>> os.getresgid()
(1000, 1000, 1000)
```

L'identifiant 0 est celui de root, 1000 celui du premier utilisateur créé (lors de l'installation du système) pour Unix.

Si l'on ne veut pas que l'application puisse être lancée par root, on peut faire :

```
>>> if os.getuid() == 0:
...     print('Ne doit pas être lancé avec root...')
```

Enfin, il est possible de retrouver le terminal contrôlant le processus :

```
>>> os.ctermid()
'/dev/tty'
```

Pour plus d'informations :

```
$ man tty
```

Pour tester les différences, la console peut être lancée en root :

```
$ sudo python3
```

2.1.4 Utilisateurs et groupes

Un système d'exploitation moderne est multi-utilisateur et permet de retrouver des informations sur les utilisateurs déclarés.

Python permet de rechercher des renseignements sur les utilisateurs en se servant des fichiers qui stockent ces informations :

```
>>> with open("/etc/passwd") as f:
...     users = [l.split(':', 6) for l in f]
... 
```

On peut ainsi afficher les données de l'utilisateur simplement :

```
>>> users[0]
['root', 'x', '0', '0', 'root', '/root', '/bin/bash\n']
```

Elles correspondent respectivement aux :

- nom d'utilisateur ;
- mot de passe encrypté (ou stocké chiffré dans un fichier séparé) ;
- identifiant de l'utilisateur ;
- identifiant de son groupe ;
- nom complet ;
- répertoire d'accueil ;
- shell au démarrage de sa session.

Avec ces données, celui qui connaît le fonctionnement de son système sait à quels niveaux il peut intervenir pour effectuer des modifications.

Voici comment obtenir l'ensemble des valeurs utilisées pour les différents utilisateurs :

```
>>> {u[6] for u in users}
{'/bin/sh\n', '/bin/false\n', '/bin/sync\n',
 '/bin/bash\n', '/usr/sbin/nologin\n'}
>>> {u[5] for u in users}
{'/home/sch', '/var/www', '/root', '/var/lib/bacula', [...]}
```

On peut utiliser les mêmes procédés pour les groupes :

```
>>> with open("/etc/group") as f:
...     groups = [l.split(':', 3) for l in f]
...
>>> groups[0]
['root', 'x', '0', '\n']
```

Le troisième élément est le numéro du groupe servant de lien avec l'utilisateur.

Voici comment mettre cette relation en évidence :

```
>>> guser = {u[3]: u[0] for u in users}
>>> user_group = [(guser.get(g[2]), g[0]) for g in groups]
```

On voit donc en peu d'exemples comment utiliser la puissance des types de Python. Le bon type pour la bonne utilisation.

Voici un script testant l'existence des éléments caractéristiques d'un utilisateur, nom d'utilisateur, le nom de groupe associé et son dossier personnel à l'emplacement par défaut :

```
>>> for username in ['sch', 'existepas']:
...     if username in (u[0] for u in users):
...         print("L'utilisateur %s existe déjà" % username)
...     if username in (g[0] for g in groups):
...         print("Le groupe %s existe déjà" % username)
...         home = '/home/%s' % username
...         if os.path.exists(home):
...             print('Le dossier %s existe déjà' % username)
...
L'utilisateur sch existe déjà
Le groupe sch existe déjà
Le dossier sch existe déjà
```

Enfin, pour terminer, les utilisateurs courants du système (pas les utilisateurs Apache, Bacula ou autre) ont un identifiant compris entre certaines bornes :

```
>>> max_user_id = max([id for id in (int(u[2]) for u in users) if
1000 < id < 19999])
>>> max_group_id = max([id for id in (int(g[2]) for g in groups)
if 1000 < id < 19999])
```

Voici les résultats pour ma machine qui ne contient que deux comptes d'utilisateurs courants :

```
>>> max_user_id, max_group_id
(1001, 1001)
```

2.1.5 Constantes pour le système de fichiers

Le système d'exploitation définit certaines caractéristiques par rapport au système de fichiers. Il s'agit de la notation du répertoire courant, du répertoire parent, du séparateur de répertoire (il peut y en avoir un second), du séparateur d'extensions, de la séparation entre chemins lorsqu'ils sont écrits l'un après l'autre, et le séparateur qui définit le changement de ligne :

```
>>> os.curdir, os.pardir
('.', '..')
>>> os.sep, os.altsep, os.extsep, os.pathsep, os.linesep
('/', None, '.', ':', '\n')
```

Chaque système définit également un chemin par défaut (liste de répertoires séparés par le séparateur `os.pathsep`) et un chemin vers une interface nulle :

```
>>> os.defpath
```